JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 6, pp. 39-51, June 2024
https://doi.org/10.9708/jksci.2024.29.06.039

# A Technique for Accurate Detection of Container Attacks with eBPF and AdaBoost

Hyeonseok Shin*, Minjung Jo*, Hosang Yoo*, Yongwon Lee*, Byungchul Tak*

*Student, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea
*Student, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea
*Student, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea
*Student, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea
*Professor, School of Computer Science and Engineering, Kyungpook National University, Daegu, Korea

## [Abstract]

This paper proposes a novel approach to enhance the security of container-based systems by analyzing system calls to dynamically detect race conditions without modifying the kernel. Container escape attacks allow attackers to break out of a container's isolation and access other systems, utilizing vulnerabilities such as race conditions that can occur in parallel computing environments. To effectively detect and defend against such attacks, this study utilizes eBPF to observe system call patterns during attack attempts and employs a AdaBoost model to detect them. For this purpose, system calls invoked during the attacks such as Dirty COW and Dirty Cred from popular applications such as MongoDB, PostgreSQL, and Redis, were used as training data. The experimental results show that this method achieved a precision of 99.55%, a recall of 99.68%, and an F1-score of 99.62%, with the system overhead of 8%.

▸ **Key words**: eBPF, System Call, Dirty COW, Dirty Cred, AdaBoost

## [요 약]

이 논문은 컨테이너 기반의 시스템 보안 강화를 목표로, 커널을 수정하지 않고 시스템콜을 분석하여 경쟁 상태를 동적으로 감지하는 새로운 방법을 제시한다. 컨테이너 탈출 공격은 공격자가 컨테이너의 격리를 벗어나 다른 시스템에 접근할 수 있게 하는데, 이 중 경쟁 상태 기반의 공격은 병렬 컴퓨팅 환경에서 발생할 수 있는 보안 취약점을 이용한다. 이러한 공격을 효과적으로 감지하고 방어하기 위해, 본 연구에서는 eBPF를 활용하여 공격 시 발생하는 시스템콜 패턴을 관찰하고, AdaBoost 모델을 사용하여 공격 프로세스와 정상 프로세스를 구분하는 방법을 개발하였다. 이를 위해 Dirty COW, Dirty Cred와 같은 공격과 MongoDB, PostgreSQL, Redis와 같은 일반 컨테이너 사용 사례에서 발생하는 시스템콜을 분석하여 학습 데이터로 활용하였다. 실험 결과, 이 방법은 99.55%의 Precision, 99.68%의 Recall 그리고 99.62%의 F1-score를 달성했으며, 이로 인한 시스템 오버헤드는 약 8%로 나타났다.

▸ **주제어**: eBPF, 시스템콜, Dirty COW, Dirty Cred, AdaBoost

• First Author: Hyeonseok Shin, Minjung Jo, Corresponding Author: Byungchul Tak
*Hyeonseok Shin (quf265@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
*Minjung Jo (jmjknu5400@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
*Hosang Yoo (yhs2739@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
*Yongwon Lee (yongdev@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
*Byungchul Tak (bctak@knu.ac.kr), School of Computer Science and Engineering, Kyungpook National University
• Received: 2024. 03. 28, Revised: 2024. 05. 30, Accepted: 2024. 06. 19.

Copyright © 2024 The Korea Society of Computer and Information
http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

www.kci.go.kr

## I. Introduction

기업의 IT 인프라에 컨테이너 기술을 효과적으로 활용함으로써 비용 절감, 신속한 서비스 제공, 유연한 관리 등 다양한 이점을 얻을 수 있다. 이는 빠른 의사 결정을 가능하게 하며, 기업의 이윤 확대에도 크게 기여하는 사례들이 있어 금융, 제조, 의료, 교육 등 다양한 산업 전반에서 컨테이너의 활용이 급격하게 늘어나고 있다[1]. 컨테이너 활용이 증가함에 따라 컨테이너의 보안성에 대한 중요도도 함께 높아지고 있다. 컨테이너가 하나의 호스트 커널을 공유하는 구조 때문에, 한 컨테이너의 보안 위협이 다른 컨테이너로 확산될 위험이 있다. 이는 가상 머신(Virtual Machine)에 비해 보안 공격으로 인한 피해가 더 클 수 있음을 의미한다. 공격자는 컨테이너 내의 커널 또는 호스트 커널의 보안 취약점을 악용하여 컨테이너 내에서 탈출하고 호스트와 다른 컨테이너들까지 접근하는 심각한 보안 위협을 초래할 수 있다. 보안 취약점의 대표적인 예로는 Dirty Copy-On-Write(Dirty COW)[2], Dirty Cred[3]가 존재한다.

경쟁 상태(Race condition) 기반의 공격은 병렬 컴퓨팅 환경에서 발생하는, 프로세스 또는 스레드가 공유 자원에 대해 동시에 접근하려고 시도할 때 발생하는 오류를 이용한 공격 방식이다. 이러한 경쟁 상태는 시스템의 예측 불가능한 동작을 초래할 수 있으며, 특히 보안 취약점을 악용하는 데 사용될 수 있다. 컨테이너 탈출(Container Escape) 공격은 이러한 경쟁 상태 기반 공격을 활용할 수 있는 한 예로, 공격자가 경쟁 상태 기반 공격을 이용하여 컨테이너의 격리된 환경을 벗어나 호스트 시스템이나 다른 컨테이너에 접근할 수 있게 하는 기법이다. 특히, 리눅스 운영체제의 Dirty COW 및 Dirty Cred와 같은 대표적인 경쟁 상태 기반 공격을 통해 컨테이너 탈출을 시도하는 예가 존재한다[4][5]. Dirty COW 취약점은 리눅스 커널의 Copy-On-Write 메커니즘에서 발생하는 경쟁 상태를 악용하여, 낮은 권한의 사용자가 읽기 전용 파일을 수정할 수 있게 함으로써 시스템의 보안 구조를 근본적으로 훼손시키는 공격이다. 이와 유사하게, Dirty Cred 취약점은 커널 자격 증명을 권한이 있는 자격 증명과 교환하여 권한을 높이는 공격으로, 경쟁 상태를 이용해 충분한 time window를 확보하는 방법을 사용한다.

기존 연구에서는 동적 탐지 기반의 시스템콜을 추적하여 경쟁 상태를 탐지하는 접근 방식은 아직 시도된 바 없다. 기존의 방식에서는 커널의 보안 취약점 발견 시, 커널 코드를 수정하여 원천 차단하는 방식을 사용하였으나, 이러한 방법은 취약점을 발견하고 수정하는 데 많은 시간과 복잡성을 요구하였다. 실제로 Dirty COW는 2007년 커밋된 리눅스 커널 2.x부터 4.8.3 버전에서 발견될 수 있었으나, 발견 및 수정되기 전까지 오랜 기간 동안 패치되지 않았다. 심지어 수정된 커널 버전을 배포한다 할지라도 Linux.BtcMine.174[6] 사례처럼 여전히 수정된 커널 버전을 반영하지 못하는 시스템을 활용하여 가상 화폐 채굴에 사용하는 등의 사례도 발생하였다. 이러한 커널 수정 방법은 취약점을 식별하는 데 시간이 많이 소요되며, 수정 사항이 모든 시스템에 적용되기까지 많은 시간이 필요하다는 큰 단점을 가지고 있다.

호스트와 커널을 공유하는 컨테이너의 취약점을 막기 위해 시스템콜 필터링[7][8][9][10][11][12][13], 컨테이너 커널 접근 제한[14], 추가 계층 삽입 등의 방법이 있다[15][16][17]. 300여 개의 시스템콜 중에서 특별한 경우를 제외하고 요청할 시스템콜의 종류가 제한적이기에 시스템콜 필터링 및 커널 접근 제한 방법은 효과적인 방안으로 평가된다. 그러나 필요한 시스템콜과 접근 범위를 명시하는 프로파일을 작성 및 유지의 어려움과 변경 사항이 발생 시 프로파일을 갱신해야 하는 추가적인 관리가 필요하다[18]. 이를 해결하기 위해 동적 혹은 정적 분석을 활용하여 보안 프로파일을 자동으로 생성하는 방법이 연구되었다[10][12][19]. 그 외에도 계층 삽입은 큰 오버헤드가 있다는 한계점이 있기 때문에 추가적인 계층이나 별도의 커널 수정 없이 동적으로 탐지 가능한 접근이 필요하다.

본 논문에서는 컨테이너 탈출이 경쟁 상태 기반 공격에 의해 발생하는지의 여부를 파악하기 위해, 커널 내부 동작인 시스템콜을 관찰함으로써 동적으로 경쟁 상태 기반 공격을 탐지하는 기법을 제안한다. 경쟁 상태는 여러 프로세스나 스레드가 동시에 공유 자원에 접근해야 발생하며, 이러한 상황을 유발하기 위하여 적절한 타이밍에 접근하는 것이 필수적이다. 그러나, 이러한 타이밍을 찾는 일은 복잡한 시스템 환경에서 매우 어려운 일이며, 이에 따라 경쟁 상태를 유발하기 위한 시도가 반복적으로 이루어진다. 이 과정에서, 관련 프로세스들은 동일한 시스템콜을 반복적으로 호출하게 되는데 이는 일반적인 프로세스의 행위와는 상이하다. 일반적으로, 하나의 프로세스가 단일 시스템콜을 과도하게 호출하는 경우는 드물기 때문에, 이러한 행위 패턴을 경쟁 상태의 징후로 판단할 수 있는 근거가 된다. 따라서, 본 연구에서 제안하는 메커니즘은 이러한 비정상적인 시스템콜 패턴을 실시간으로 탐지하여 경쟁 상태 기반의 공격을 감지하는 것이다.

www.kci.go.kr

커널 내부에서 발생하는 다양한 이벤트들을 관측하는 기술은 시스템의 효율성 및 보안에 있어 중대한 역할을 수행한다. 본 연구에서는 커널 이벤트 관측을 위해 extended Berkeley Packet Filter(eBPF)를 활용한다. eBPF는 기존의 커널 이벤트 관측 도구들과 비교할 때, 몇 가지 결정적인 이점을 가진다. 첫째로, eBPF는 커널을 다시 빌드하거나 수정할 필요 없이 사용자가 원하는 로직을 커널 내부에 직접 프로그래밍할 수 있게 해주는 강력한 기능을 제공한다. 둘째로, eBPF 프로그램은 커널과 사용자 공간 사이의 상호 작용을 최적화하여, 데이터 처리 및 응답 시간을 개선함으로써 신속한 공격 감지를 가능하게 한다. 마지막으로, 커널 영역에서 커널 로그 분석을 수행할 수 있어 관측과 더불어 실시간에 가까운 분석이 가능하다. 이는 경쟁 상태 기반 공격과 같이 빠르게 변화하는 시스템 상태를 감지하는 데 필수적이다. 본 연구에서는 eBPF를 이용하여 시스템콜이 호출될 때 발생하는 sys_enter 이벤트를 감지하고, 이를 기반으로 프로세스별 시스템콜 호출 데이터를 수집한다. 수집된 데이터에서 특정 프로세스의 시스템콜 총 호출 수, 시스템콜 인자의 유사도, 시스템콜 종류의 유사도, 그리고 호출된 시스템콜의 종류 수 등 다양한 특징을 추출한다. 추출한 데이터의 특징을 이용하기 위해 우리는 머신러닝 기법 중 Linear classification, Support Vector Machine(SVM), Decision tree 기반의 앙상블(Ensemble) 기법인 Random Forest, Adaptive Boosting(AdaBoost) 모델을 학습에 적용하여 공격 프로세스와 정상 프로세스를 구분한다.

본 논문은 다음과 같은 기여를 한다.

- 경쟁 상태 기반의 공격을 활용한 컨테이너 탈출 시도를 감지하기 위해, eBPF와 머신러닝 알고리즘을 활용하여 프로세스의 시스템콜을 실시간으로 관측, 분석하고, 이를 통해 경쟁 상태 기반 공격을 동적으로 감지하는 기법을 제안한다.
- 경쟁 상태 기반 공격이 호출하는 시스템콜의 정량적 수치와 일반 프로세스의 정량적 수치를 비교 분석하여 통계적 차이를 밝혔다. 우리의 조사에 따르면, 경쟁 상태 기반 공격은 전체 시스템콜 중에서 특정 시스템콜의 비중, 종류의 유사도 및 인자의 유사도에서 일반 프로세스보다 높은 수치를 기록하였다.

# II. Background

본 장에서는 연구의 기반이 되는 시스템에 대한 배경지식에 대해 설명한다.

## 1. eBPF

eBPF는 리눅스 커널 내에서 사용자 정의 코드를 안전하고 효율적으로 실행할 수 있게 해주는 강력한 기술이다. eBPF는 1992년에 개발된 BPF를 기반으로, 네트워킹, 보안, 추적, 성능 분석 등 다양한 분야에서 활용되고 있다. 초기에는 네트워크 트래픽 분석 및 필터링하기 위한 목적으로 개발되었지만, 이후 다양한 확장과 함께 리눅스 커널의 거의 모든 이벤트를 모니터링하고, 추적하며, 커널 함수를 최적화하는 등 다양한 용도로 확장되고 있다.

eBPF는 리눅스 커널 내의 다양한 probe객체를 통해, 이벤트를 감지하고 사용자가 작성한 eBPF 코드를 실행한다. eBPF 프로그램은 자체 검증기를 통해 커널에 악영향을 주지 않는 안전한 헬퍼 함수로 동작하기 때문에 커널에 악영향을 주지 않으면서도, 성능과 유연성 측면에서 많은 이점을 제공한다. eBPF는 Security, Networking, Tracing, Observability 등의 다양한 사용 사례를 통해 시스템의 보안성을 강화하고, 효율적인 네트워크 트래픽 처리, 시스템 및 응용 프로그램의 런타임 동작 추적, 시스템 메트릭 수집이 가능하다.

리눅스 기반 컨테이너의 경우, eBPF를 통한 모니터링은 성능을 유지하고 효율적인 운영에 필수적이다. 이는 운영체제가 시스템 리소스를 어떻게 활용하는지, 어떤 프로세스가 가장 많은 리소스를 소비하는지 등의 중요한 정보를 제공하여 성능 최적화 및 효율적인 리소스 관리를 위한 전략을 수립할 수 있게 한다. 또한, eBPF는 성능 병목 현상이나 이상 징후를 조기에 발견하고, 원인을 파악하여 적절한 대응을 할 수 있게 도와주며, 실시간으로 성능 데이터를 제공하여 문제 발생 시 빠른 대응을 가능하게 한다. 이를 통해 컨테이너의 성능을 유지하고 시스템의 안정성을 높이며 비용 효율성을 개선할 수 있다[20][21][22].

## 2. Dirty COW

CVE-2016-5195로도 알려진 Dirty COW는 리눅스 커널에서 발견된 권한 상승(Privilege Escalation) 취약점 중 하나이다[2]. 해당 취약점은 2007년부터 2016년까지 리눅스 커널에 존재했으며 현재는 패치된 상태이다. Dirty COW 공격은 경쟁 상태를 이용하여 Copy-on-Write 과정에서 발생하는 오류를 통해 읽기 권한이 있는 파일에 쓰기 작업을 가능하게 한다.

www.kci.go.kr

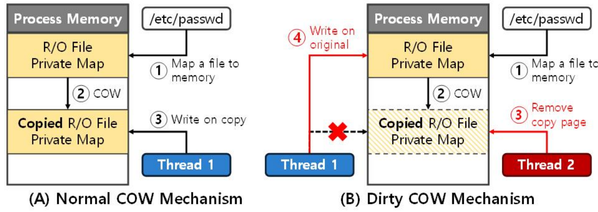


Fig. 1. Dirty COW echanism

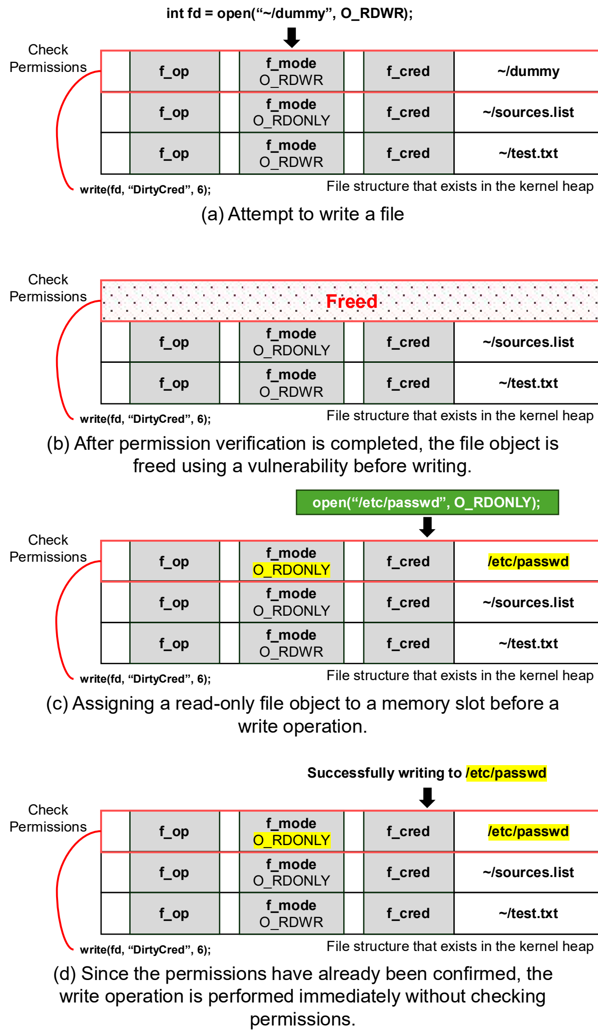


Fig. 2. How Dirty Cred works

Fig. 1의 (A)는 정상적인 Copy-on-Write의 메커니즘을 보여준다. 반면, (B)의 경우 Dirty COW 상황을 나타낸다. 여기서는 한 스레드가 복사본에 대한 쓰기 작업을 진행하기 전에 다른 스레드가 먼저 실행되어 해당 복사본을 삭제하게 될 경우, 원래의 스레드는 원본을 복사본으로 오인하여 수정해야 할 대상이 아닌 원본을 수정하는 문제가 발생한다.

## 3. Dirty Cred

Dirty Cred는 커널의 취약한 설계를 이용하여 사용자 권한을 관리자 권한으로 승격시키는 공격 기법이다[3]. 이 기법에서는 Use-After-Free(UAF)와 Double Free(DF) 같은 대표적인 취약점이 사용된다.

UAF 취약점은 동적 메모리를 잘못 사용할 때 발생한다. 메모리를 해제한 후에도 프로그램이 해당 메모리에 대한 포인터를 제거하지 않으면 공격자가 해당 오류를 이용하여 프로그램을 공격할 기회를 얻게 된다. DF 취약점의 경우 동일한 메모리 위치에 대한 해제를 두 번 수행함으로써 발생하는 가장 일반적인 메모리 손상 오류이다. 해당 오류를 통해 공격자는 메모리 손상, 임의 주소 쓰기 및 읽기, 임의 코드 실행 등 다양한 수단으로 활용될 수 있다.

Fig. 2에서는 Dirty Cred의 동작 방식을 상세히 설명하고 있다. Dirty Cred는 낮은 권한을 가진 파일 객체를 커널에 할당한다. 할당된 파일 객체는 쓰기 권한을 가지며, 커널은 쓰기 작업이 진행되는 동안 해당 파일 객체의 쓰기 권한을 확인한다. 권한 확인이 완료된 후, Dirty Cred는 쓰기 작업을 일시적으로 중단하고, 파일 시스템의 설계를 이용하여 time window를 늘리기 위한 작업을 진행한다. 특히, 경쟁 상태를 활용해 쓰기 작업 중 time window를 확장하고 객체 스왑을 수행한다. 이와 같은 방법은 취약점을 이용하면서도 시스템 안정성에 문제를 일으키지 않으면서 필요 충분한 시간을 확보할 수 있도록 한다. 충분한 시간을 확보한 Dirty Cred는 UAF 혹은 DF와 같은 커널 취약점을 이용하여 작업 중인 파일 객체 영역을 해제하는 작업을 수행한다. 해제된 영역은 쓰기 권한이 유지되며, 읽기 권한만 부여된 높은 권한의 파일 객체를 할당받아 쓰기 작업을 재개한다. 쓰기 권한 검증이 이미 완료된 상태에서 높은 권한의 파일 객체로 교체됨으로써, Dirty Cred는 파일 수정 권한을 획득하게 된다. 이 과정을 통해 Dirty Cred는 시스템의 취약점을 악용하여 높은 권한으로의 권한 상승을 성공시킨다.

## 4. Machine Learning & AdaBoost

머신러닝은 과거부터 딥러닝이 나온 현대에까지 꾸준히 사용되는 학습 기법이다. SVM, KNN과 같은 고전적인 머신러닝 기법은 해석 가능성 면에서 딥러닝의 약점을 보완하는 장점을 가지고 있다. 딥러닝은 판단의 근거를 파악하기 어려운 반면, 머신러닝의 경우 판단의 근거를 상대적으로 쉽게 제공한다. 또한, 딥러닝을 사용하기 위해 막대한 양의 데이터가 필요하다. 하지만 정상적으로 동작하는 exploit code를 많이 확보하는데 어려움이 있기 때문에

www.kci.go.kr

실제 적용에는 제한적이다. 반면, 머신러닝은 Linear classification, SVM과 같은 기본적인 모델과 weak learner로 결합한 Ensemble 모델과 같이 다양한 모델들이 존재하는데 그중에서 특히 Ensemble model은 저차원의 적은 데이터로도 딥러닝과 큰 차이가 나지 않는 정확도를 달성하며, 판단에 근거를 쉽게 제시할 수 있어 공격 탐지 시스템에 적합하다.

본 논문에서 사용한 머신러닝 모델은 4가지로 Linear classification, SVM, Random Forest, AdaBoost이다. Linear classification, SVM은 기본적인 머신러닝 모델로 동작이 단순하고 해석이 쉽다는 장점이 있다. Ensemble model은 여러 weak learner를 사용하여 강력한 모델을 구축하는 기법으로, voting, bagging, boosting 등의 방법이 있고 대표적으로, boosting 기법에는 AdaBoost, bagging 기법에는 Random Forest가 있다. 그중에서 Boosting은 weak learner를 순서대로 학습시키며 추가해 나가고 최종적으로 만들어진 weak learner들을 이용해서 강력한 모델을 구축하는 기법이다. 학습기를 추가해나갈 때, 앞에서 예측한 학습기에서 틀린 부분에 가중치를 더 주고 다음 학습기는 가중치가 높아진 틀린 부분에 대해서 학습을 잘하게 된다. 이를 통해서 모델의 성능을 향상시킨다. Bagging 기법은 훈련데이터를 무작위로 복원 추출하여 여러 weak learner를 병렬적으로 학습시키는 기법이다. 각각의 weak learner가 판단을 수행하고 그 결과를 합쳐서 최종 판단을 하게 된다. Bagging 기법은 여러 weak learner를 사용함으로써 학습 데이터에 대한 편향을 낮출 수 있어서 강력한 성능을 발휘한다.

## III. Related Work

컨테이너는 OS 커널을 공유하므로 애플리케이션당 전체 OS가 필요하지 않아 컨테이너 크기를 줄일 수 있다. 하지만 OS 커널을 공유하는 방식은 여러 보안 취약점에 노출될 수 있으며, 이를 방어하기 위한 다양한 연구가 진행되었다. 본 장에서는 컨테이너의 보안 취약점 대응 및 커널 보안을 위해 제안된 다양한 연구를 소개한다.

### 1. Sysfilter

응용프로그램은 시스템콜 API를 통해 OS 커널과 상호작용하며 유용한 작업을 수행한다. 응용프로그램은 커널이 제공하는 300여 개 이상의 시스템콜 중 일부만 필요하지만, OS 커널은 전체 시스템콜 셋에 대한 완전하고 제한 없는 접근을 제공한다. 이는 최소 권한 원칙을 위반하며, 공격자는 이를 악용하여 권한 상승의 가능성을 높인다. 이를 해결하기 위해 Sysfilter는 최소 권한 원칙을 강화하고 공격 표면을 줄여 실제 위협을 완화하는 방법을 제안했다[8].

Sysfilter는 대상 응용프로그램의 이진 파일을 입력받아 함수 호출 그래프를 생성하고, 이를 통해 함수 간의 호출 관계를 나타내어 응용프로그램이 필요로 하는 시스템콜 셋을 추출한다. 추출한 시스템콜 셋은 Seccomp-BPF 프로그램(필터)으로 변환되며, 해당 필터는 커널이 시스템콜을 호출할 때마다 강제로 실행되어 시스템콜의 접근을 허용 혹은 거부를 결정한다. 이러한 방식은 특정 시스템콜을 차단하는 데 중점을 둔다. 그러나 이 방식은 커널 취약점에 대해 완벽한 방어를 제공하지 못하며, Sysfilter의 효과는 이진 파일 수준에서만 평가된다는 한계점이 있다.

### 2. Confine

컨테이너는 주로 하나의 애플리케이션이나 서비스 실행에 특화되어 있으나, 메인 프로그램을 실행하기 전에 다양한 유틸리티와 지원 프로그램을 호출한다. 이 과정에서 악의적인 프로그램이 포함된 컨테이너는 커널의 취약점을 이용하여 동일한 호스트를 공유하는 다른 컨테이너에 악영향을 미칠 수 있다. 이러한 문제를 해결하기 위해 컨테이너의 시스템콜 접근을 제한하여 보안을 강화하는 자동화 기술인 Confine이 제안되었다. 이는 컨테이너가 실제로 필요한 시스템콜만 접근할 수 있도록 하여 커널 인터페이스의 취약한 부분을 보호한다[10].

Confine은 컨테이너의 생명 주기와 실행되는 애플리케이션을 분석함으로써 필요한 시스템콜을 추출한다. 이 과정은 동적 분석과 정적 분석을 모두 포함하며, 라이브러리 소스 코드 분석을 통해 시스템콜과 함수 간의 매핑을 도출하고 필요한 경우 binary code disassembly를 사용한다. 추출된 시스템콜 리스트를 바탕으로 사용자는 Seccomp 필터를 사용하여 컨테이너의 시스템콜 정책을 구성할 수 있으며, 이는 주로 거부 목록 형태로 사용된다. 추출된 시스템콜 리스트를 기반으로 자동화된 스크립트를 사용하여 금지된 시스템콜 리스트를 도출하고 Seccomp 프로필을 구성하여 최종적으로 컨테이너의 시스템콜 정책을 완성한다. 그러나 Confine은 Docker 위주로 제작되어 다른 컨테이너 런타임에 완벽하게 적용하기 어렵다는 한계가 있다. 또한, 프로그램이 시스템콜을 직접적으로 호출하는 것을 대비하기 위해 binary code disassembly를 진행하는데, 이 과정은 x86-64 CPU 아키텍처만 지원한다는 제한이 있다.

www.kci.go.kr

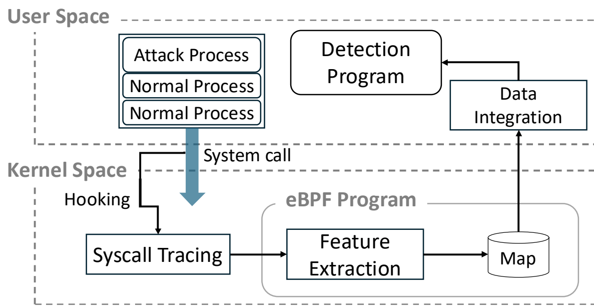


Fig. 3. Race condition detection framework

### 3. bpfbox

앞서 설명한 Confine처럼 컨테이너의 보안을 위해 제안된 또 다른 방식으로 bpfbox가 있다. 기존의 프로세스 제한 방법들은 namespace, cgroup, SELinux, AppArmor 등을 사용하여 복잡하게 구현되었으나, 이러한 방식은 유연한 대처가 어렵다. bpfbox는 eBPF 방식을 활용하여 보다 간단하고 효율적으로 프로세스 제한을 수행하는 방법을 제안하였다. bpfbox는 시스템콜, 사용자 공간 함수, LSM 후크를 통해 프로세스 제한을 구현하며, 약 2000줄 미만의 코드로 구성되어 있다[9].

bpfbox의 구조는 사용자 영역 데몬, eBPF 프로그램 모음, 그리고 정책 정보를 저장하는 여러 맵으로 이루어져 있다. bpfbox는 정책 파일을 파싱하고 eBPF 코드를 컴파일하여 커널에 로드하며, eBPF 맵을 통해 실행 중인 eBPF 코드와 상호작용한다. 정책은 사용자 정의 언어로 작성되며, 특정 작업을 allowed, audited, logged, tainted로 구분한다. 정책은 하나의 실행 파일당 하나로, 루트 제어 전용 디렉토리에 저장된다. 정책 로드 시, 정책 구조는 eBPF 프로그램을 통해 커널에 기록된다.

bpfbox는 파일 시스템 접근, 프로세스 간 통신, 네트워크 소켓 등을 포함하는 여러 LSM 프로브를 구현하며, eBPF 프로그램은 프로세스 상태 확인, 정책 키 결정, 정책 조회, 오염 상태 확인, 접근 허용 및 거부, 감사 이벤트 제출의 6단계로 구성된다. bpfbox는 임시 파일 처리에 한계가 있지만, 이를 극복하기 위해 동적으로 임시 규칙을 생성한다. bpfbox는 bcc 도구를 사용해 구현되었으며, 시스템콜, 네트워크 연결, 함수 제한 등 다양한 제한을 가능하게 한다. 그러나 bpfbox는 실행 전에 시작된 프로세스에는 적용할 수 없고, eBPF 프로그램의 재귀 함수 호출 및 반복문 사용에 제약이 있다.

## IV. Detection Model Implementation

본 절에서는 시스템콜 로그를 트레이싱하고 수집한 데이터를 전처리하여 학습 데이터로 사용하는 과정을 상세히 설명한다.

### 1. Overview of Race Condition Attack Detection

Fig. 3은 경쟁 상태 기반 공격 감지 프레임워크이다. 이 프레임워크는 크게 2가지 핵심 과정으로 구성된다. 첫 번째 과정은 커널 영역에서 스레드에 의해 호출되는 시스템콜을 추적하는 과정이다. 이 과정에서 eBPF 프로그램은 여러 스레드들이 시스템콜을 호출할 때마다 이를 후킹하여 추출된 정보를 eBPF 프로그램 내의 해시(Hash) 자료형 저장소인 맵에 해당 정보들을 저장한다. 두 번째 과정에서는 사용자 영역에서 eBPF 프로그램에 의해 추출된 특징 데이터를 이용하여 스레드 단위 데이터를 프로세스 단위 데이터로 통합한다. 통합된 데이터는 머신러닝 모델의 학습에 사용되며, 최종적으로 학습된 모델은 시스템이 실시간으로 호출하는 시스템콜 데이터들을 평가하여 공격 프로세스를 감지한다.

커널 영역에서 시스템 로그 수집과 사용자 영역에서 공격 탐지의 분리는 다수의 이점을 제공한다. 시스템콜 로그 수집은 커널 영역에서 진행하기 때문에 컨텍스트 스위치에 의한 오버헤드가 없다. 일반적으로 사용자 영역에서 로그 수집 시, 로그 수집 프로그램은 지속적인 CPU 점유와 함께 로그를 수집해야 하며 이 과정에서 반복적인 컨텍스트 스위치로 인한 상당한 오버헤드가 발생한다. 그러나, eBPF 프로그램을 활용할 경우, 이러한 오버헤드는 발생하지 않는다. 사용자 영역에서 시스템콜 로그를 일정 주기적으로 가져가므로 실시간으로 시스템콜 로그를 가져가지 않아서 cpu 오버헤드가 적고 사용자 영역에서 시스템콜 로그를 가져가더라도 커널 영역에서는 계속해서 시스템콜 로그를 수집하고 있으므로 로그 손실이 없다. 시스템콜 로그를 사용자 영역에서 주기적으로 추출함으로써 실시간 로그 추출이 필요 없어 CPU 오버헤드가 감소하며, 커널 영역에서는 지속적인 로그 수집이 이루어지므로 로그 손실이 발생하지 않는다. 이러한 메커니즘을 통해 컨테이너 공격 탐지를 위한 정확한 주기적 검사가 가능하다.

커널 영역에서 시스템콜 로그 수집과 공격 탐지를 동시에 수행할 경우, 탐지 속도가 향상될 수 있으나, 이는 커널 내 연산량의 증가를 초래하고, 기계 학습 기법에 비해 탐지 정확도가 감소할 수 있다. 이는 이상 탐지에서 크게 우려되는 거짓 양성(FP, False Positive)의 증가로 이어질

www.kci.go.kr

수 있다. 반면, 사용자 영역에서의 검사는 약간의 지연은 있지만, 실시간 탐지를 거의 보장한다. 공격자는 검사를 하는 time window의 시작점과 끝점을 모르기 때문에 공격은 해당 time window보다 훨씬 짧은 시간 안에 이루어져야 한다. 또한, time window가 짧아 공격이 성공하더라도, 사후 탐지를 통한 신속한 대응으로 피해를 최소화할 수 있다.

## 2. System Call Tracing

시스템콜 호출 시, 커널에서는 sys_enter 이벤트가 발생하며, 이를 eBPF 프로그램이 캐치(catch)하여 시스템콜 실행을 일시적으로 중지시키고, eBPF 코드를 실행시킨다. eBPF 코드는 호출된 시스템콜의 종류, 사용된 인자 정보를 파악할 수 있으며, eBPF 헬퍼(helper) 함수를 통해 해당 시스템콜을 호출한 프로세스 ID와 스레드 ID를 획득할 수 있다. 이렇게 수집된 정보는 해시 기반의 맵에 저장되어, eBPF 코드 실행 후에도 유지되며 사용자 영역에서 읽을 수 있게 된다.

## 3. Feature Extraction from System Call Logs

eBPF를 활용하면 커널 내에서 발생하는 시스템콜 로그를 실시간으로 수집할 수 있으며, 이를 통해 호출 인자, 호출한 스레드 ID, 프로세스 ID 등의 정보를 획득할 수 있다. 본 연구는 이러한 정보를 기반으로 학습 데이터로 사용할 특징 추출을 진행하였다. 구체적으로 100ms 동안 발생한 시스템콜 총 호출 개수, 발생한 시스템콜 호출의 종류, 동일 시스템콜의 연속적 사용 여부, 시스템콜 호출 시 사용된 인자의 연속적 일치 여부를 주요 특성으로 선정하여, 이를 머신러닝 학습 데이터로 활용하였다.

시스템콜의 총 호출 개수는 스레드 별로 집계된 시스템콜 호출 수를 의미한다. 호출된 시스템콜의 종류는 100ms 동안 호출된 시스템콜의 종류로, 예를 들어 3가지의 시스템콜을 호출하였다면, 해당 값은 3이 된다. 동일 시스템콜의 연속 호출은 같은 스레드 내에서 동일한 시스템콜이 연속적으로 사용된 횟수를 나타낸다. 시스템콜 호출 시 사용된 인자의 연속적 동일성 여부는 스레드가 시스템콜을 호출할 때 같은 인자를 사용하여 호출하는 정도를 나타낸 것으로 하나의 스레드가 시스템콜을 호출할 때 직전에 시스템콜 호출에 사용한 인자와 동일한 인자를 하나라도 사용할 때 값이 올라간다. 이러한 특징들은 향후에 프로세스 단위로 합쳐져서 feature로 사용된다.

일반적으로 경쟁 상태를 일으키려면 빠르게 같은 시스템콜을 호출해야 하기 때문에 다음과 같은 특징이 나타난다. 1)시스템콜 호출 수가 많고 2)적은 종류의 시스템콜을 호출하고 3)동일한 시스템콜을 연속적으로 많이 호출하고 4)비슷한 인자를 사용한다. Feature에 관한 내용은 5.1장에서 추가로 논의한다.

## 4. Data Preprocessing

경쟁 상태는 여러 스레드가 동일 자원에 동시 접근 할 때 발생하며, 이로 인해 단일 스레드의 시스템콜 호출 정보만으로 프로세스 내의 경쟁 상태를 정확히 판단하기 어렵다. 스레드 단위로 수집된 시스템콜 로그 데이터는 바로 사용할 수 있는 형태가 아니며, 스레드 단위가 아닌 프로세스 단위의 데이터로 통합해야 한다. 사용자 영역의 파이썬 프로그램은 100ms 간격으로 커널로부터 데이터를 수집하여 프로세스 단위로 데이터를 통합시킨다. 사용자 영역의 파이썬 프로그램은 시스템콜 로그 내의 프로세스 ID를 비교하여 동일한 프로세스 ID를 가진 데이터를 결합하여 새로운 로그를 생성한다. 커널 영역에서 수집된 시스템콜 로그에는 프로세스 ID, 호출 횟수, 시스템콜 번호 등의 정보가 포함되어 있다. 이러한 시스템콜 로그는 다음과 같이 나타낼 수 있다.

$$T = \{PID, SYSCALL_COUNT, \ldots\} \tag{1}$$

사용자 영역의 파이썬 프로그램은 로그 데이터에 있는 프로세스 ID를 확인하여 같은 프로세스 ID를 가진 로그를 하나로 모아 각각의 정보를 합치고 조합하여 프로세스 단위의 새로운 하나의 로그를 만들어 낸다. 새로운 프로세스 단위의 로그를 $P$라고 하고 $n$개의 $T$를 $\{T_n\}$라고 나타내면 $P$는 다음과 같이 나타낼 수 있다. $P$의 구체적인 항목은 5장 1절에서 자세히 다루고 있다.

$$P = \{T_n\} \tag{2}$$

프로세스 단위로 데이터를 통합할 때의 시스템콜 호출 총 개수는 스레드 단위로 있는 시스템콜 호출 총 개수를 합한다. 아래와 같이 각각의 $T$의 시스템콜 호출 수 합한 값이 $P$의 시스템콜 총 호출 수가 된다. 시스템콜 호출 수를 $s$로 나타내고 시스템콜 총 호출 수를 $d$로 나타내면 $T_i[s]$는 $i$번째 스레드 로그의 시스템콜 호출 수 요소를 나타내고 $P[d]$는 프로세스 로그의 시스템콜 총 호출 수를 나타낸다. $T_i[s]$와 $P[d]$의 관계식은 다음과 같다.

$$P[d] = \sum_{i=1}^{n} T_i[s] \tag{3}$$

시스템콜 종류는 스레드에서 사용한 시스템콜 종류가 다를 경우 개수가 올라간다. 각각의 $T$는 시스템콜 번호를 가지고 있다. 시스템콜 번호를 $s$이라고 하고 시스템콜의

www.kci.go.kr

Table 1. Benchmark Information

| Container Image | Bench Test | Bench Tool |
|---|---|---|
| MongoDB | 32 thread, 32 Client | ycsb |
| PostgreSQL | | pgbench |
| Redis | | redis-bench |
| MySQL | 32 thread, 32 Client, 40 tables, 250 tables | sysbench |
| MariaDB | | |
| Alphine | Block Size : 1KB, 100GB | |
| Ubuntu | | |
| CentOS | | |
| Nginx | Concurrency : 9999, Request : 9999 | ab |
| httpd | | |

번호의 집합을 $S_N$, $P$의 시스템콜 종류 개수 항목을 $d$라고 할 때 $d$에 대한 식은 다음과 같다.

$$S_N = \{T_n[s]\},\ P[d] = |S_N| \tag{4}$$

예를 들어서, 첫 번째 스레드는 write 시스템콜만 사용하고 두 번째 스레드는 read 시스템콜만 사용하였다면 프로세스 단위로 통합될 때 종류값은 2가 된다.

시스템콜 호출의 연속적 사용 정도와 사용된 인자의 연속적 동일성 정도는 각각의 값을 전부 합한 후에 시스템콜 호출의 총 개수로 나누어 0부터 1 사이 값을 가지게 된다. $T$의 시스템콜 호출의 연속적 사용 정도, 사용된 인자의 연속적 동일성 정도를 $s$, $P$의 시스템콜 호출의 연속적 사용 정도, 사용된 인자의 연속적 동일성 정도를 $d$, $P$의 시스템콜 총 호출 수를 $r$이라고 할 때 다음과 같은 식을 각각 만족한다.

$$P[d] = \frac{\Sigma_{i=1}^{n} T_i[s]}{P[r]} \tag{5}$$

예를 들어서, 첫 번째 스레드의 시스템콜 호출의 연속적 사용 정도는 100이고 두 번째 스레드는 200이고 두 스레드의 시스템콜 총 호출 수가 1000이라면 값은 0.3이 된다.

### 5. Collection of Training Data

학습데이터를 수집할 때와 경쟁 상태를 감지할 때 사용하는 eBPF 코드는 1가지를 제외하고는 동일하다. 학습데이터를 수집할 때는 로그 결과에 라벨링을 추가하고 파일로 출력한다. 학습된 모델을 사용하여 경쟁 상태를 감지할 때는 로그를 학습된 모델에 넣어 결과를 출력한다. 정상적인 프로세스의 데이터를 수집하기 위하여 컨테이너 이미지 10개를 각각 벤치마크 테스트를 하여 데이터를 수집하였다. Table. 1은 벤치마크 테스트에 사용한 도구와 테스트 시 사용한 파라미터를 보여준다. 벤치마크 테스트를 실행한 후에 발생한 로그는 정상 프로세스로 라벨링하여 정상 프로세스 판단 기준으로 사용하였다. 위의 테스트는 기본적으로 전부 많은 시스템콜을 호출하였고 몇몇 프로세스는 exploit code보다 더 많은 시스템콜을 호출하였다.

공격 프로세스 데이터로는 CVE-2016-5195, CVE-2017-1000405, CVE-2021-4154, CVE-2022-2588을 사용하였고 CVE-2016-5195는 5가지 버전을 사용하였다. 벤치마크 테스트와 마찬가지로 exploit code도 실행된 후에 스스로 종료될 때까지 데이터를 수집하였다. 공격 프로세스는 실제 공격 환경을 고려하기 위해서 2가지 방식으로 데이터가 수집되었다. 하나는 기존의 코드를 수정하지 않고 데이터를 수집하였고 다른 하나는 기존의 코드를 수정하여 정상적인 프로세스와 비슷한 특징을 가지도록 바꾸어 데이터를 수집하였다. 코드를 수정할 때 기존의 공격 로직은 그대로 유지하되 추가적인 스레드를 생성하여 더미 시스템콜을 발생시키고 공격 중 주기적으로 sleep을 통해 공격을 일시 중지하도록 코드를 수정하였다. 해당 결과는 5.2장에서 기술된다.

## V. Evaluation

본 논문에서는 감지에 사용되는 각 머신러닝 모델의 성능 평가 지표로 Precision, Recall 그리고 F1-score를 사용하였으며, 시스템의 오버헤드를 측정하기 위해 Phoronix test suite[23]를 활용하여 OSBench 및 Apache HTTP Server Benchmark를 시행하였다. 평가 결과, 대부분의 모델이 90% 이상, AdaBoost는 약 99%의 Precision, Recall, F1-Score를 보였으며 평균적으로 약 8%의 오버헤드가 발생하였다.

### 1. Experimental Data Configuration

CVE 관련 exploit code 분석에는 CVE-2016-5195, CVE-2017-1000405, CVE-2021-4154, CVE-2022-2588을 사용하였다. 특히, Dirty COW(CVE-2016-5195)는 5가지 다른 버전으로 분석되었으며, Dirty Cred 또한 CVE-2021-4154와 CVE-2022- 2588 2가지 버전이 존재한다. 각각의 정상 프로세스 데이터는 Alpine Linux, MySQL, MongoDB, PostgreSQL, Redis, MariaDB, Ubuntu, CentOS, Nginx, httpd 성능을 벤치마킹할 때 호출하는 시스템콜들을 활용하였다. 획득한 데이터는 다음과 같은 열 정보를 포함한다.:

- Print_Type: 프로세스의 실행 구간을 나타내며, 100ms 단위로 측정
- PID: 프로세스의 고유 식별자(Process Identifier)

www.kci.go.kr

- Process_Name: 실행 중인 프로세스 이름
- SYSCALL_TOTAL_COUNT: 100ms 동안 호출된 시스템콜의 총 합계
- SYSCALL_KIND_SIMILAR: 연속으로 호출된 시스템콜이 같은 종류일 경우 카운트를 증가시키고, 이를 100ms 당 발생하는 시스템콜의 총 호출 수로 나눈 값
- SYSCALL_KIND: 사용된 시스템콜의 가짓수
- SYSCALL_ARGUMENT_SIMILAR: 연속으로 같은 인자값을 사용한 시스템콜이 호출 될 경우 카운트를 증가시키고, 이를 100ms 당 발생하는 시스템콜의 총 호출 수로 나눈 값
- DANGER: 프로세스의 상태를 '정상' 또는 '비정상'으로 구분

해당 정보에서 feature 데이터로는 SYSCALL_TOTAL_COUNT, SYSCALL_KIND_SIMILAR, SYSCALL_KIND, SYSCALL_ARGUMENT_SIMILAR가 활용되었으며, 정답 데이터로는 DANGER에 0또는 1을 라벨링하여 사용하였다. 이를 통해 eBPF 프로그램과 AdaBoost 모델을 이용한 시스템콜 추적 및 경쟁 상태 기반 공격 감지 방법의 유효성을 검증하였다.

## 2. Observations and Analysis of System Call Log Data

Fig. 4, 5, 6, 7은 각각 시스템콜의 총 개수, 발생한 시스템콜의 종류, 동일 시스템콜의 연속적 사용 여부, 시스템콜 호출 시 사용된 인자의 연속적 동일성 여부를 exploit code와 정상 프로세스 데이터에서 추출하여 그래프로 나타낸 것이다. 각 그림에서 왼쪽은 공격감지를 어렵게 하는 패턴을 넣은 exploit code로 데이터를 추출했을 때 결과이고 오른쪽은 원본 코드를 그대로 사용했을 때의 결과이다. 시스템콜 총 호출 수는 Fig. 4와 같이 일반적으로 exploit code의 총 호출 수가 더 많았지만 MariaDB는 exploit code와 유사하게 시스템콜을 호출을 많이하였다. 사용한 시스템콜 종류를 나타내는 Fig. 6에서 exploit code는 수정 전에는 적은 종류의 시스템콜 호출을 사용하였으나 수정 후에는 정상 프로세스와 비슷하였다.

대체적으로, 각각의 결과는 수정 전에는 비교적 정상 프로세스와 exploit code를 구분하기 쉬웠지만 수정 후에는 둘의 경계가 모호해졌다. Fig. 8, 9, 10, 11은 각 feautre에 대해서 정상 프로세스와 수정 후의 공격프로세스 데이터 구성을 나타낸 그림이다. 대체로 시스템콜 총 호출 수를 제외하면 데이터 구성에서 겹치는 부분이 많다.

Table 2. Model Results

| Model | Precision | Recall | F1-Score |
|---|---|---|---|
| AdaBoost | 99.55 | 99.68 | 99.62 |
| SVM | 88.47 | 98.82 | 93.36 |
| Random Forest | 93.54 | 97.25 | 95.36 |
| Linear classification | 91.57 | 94.67 | 93.10 |

Table 3. Overhead Measurements

| Phoronix OSBench (lower is better) | Base | Ours |
|---|---|---|
| Create Files (μs) | 22.8 | 23.46 (5.20%) |
| Create Threads (μs) | 20.81 | 22.12 (6.30%) |
| Launch Programs (μs) | 140.04 | 176.57 (26.09%) |
| Create Processes (μs) | 51.11 | 52.87 (3.44%) |
| Memory Allocations (ns) | 125.72 | 124.34 (-1.1%) |
| Phoronix Apache (higher is better) | Base | Ours |
| Requests Per Second (r/s) | 19226.08 | 16998.47 (-11.59%) |

## 3. Accuracy Assessment

본 연구에서는 4개의 머신러닝 모델을 지도 학습 분류 방식을 활용한다. 이를 통해 수정한 exploit code와 정상 프로세스 데이터를 사용한 약 7만 개의 시스템콜 데이터에 대한 학습을 수행하였으며, 모델의 성능 평가는 결과는 Table. 2와 같다. 대부분의 평가가 90%를 넘으며 이는 정상 프로세스와 공격 프로세스 사이에 시스템콜 호출 양상이 상당히 다름을 시사한다. 결과 그래프가 나타내듯이 하나의 특징이 정상 프로세스와 공격 프로세스를 정확하게 분류하기는 어렵지만 다양한 특징을 보았을 때 정상 프로세스는 다음과 같이 말할 수 있다. 정상 프로세스는 공격 프로세스에 비해 일반적으로 많은 일을 하더라도 시스템콜 호출 수가 적거나 시스템콜 호출 수가 많더라도 호출한 시스템콜 종류가 더 다양하고 개수의 분포가 평탄한 모습을 보인다. 또한, 연속적으로 같은 시스템콜을 호출하는 정도가 작고 같은 인자를 사용하여도 시스템콜을 호출하는 정도가 작다. 이처럼 정상 프로세스와 공격 프로세스의 차이를 명확하게 정의할 수 있으므로 이진 분류의 성공률은 매우 높을 수 있다. 다른 모델들도 성능이 높지만 그중에서 AdaBoost의 성능이 가장 우수하다. AdaBoost는 지속적으로 잘못 판단한 데이터를 추가로 학습하면서 오류를 줄이고 현재 학습에 사용된 특징이 4개 정도로 많지 않기 때문에 성능이 좋게 나올 수 있다. 해당 모델을 바탕으로 도출된 특성의 중요도는 Fig. 12를 통해 제시한다. Fig. 12를 통해 확인한 결과 SYSCALL_TOTAL_COUNT 특성은 0.36의 가장 높은 중요도를 기록하였으며, 이는 시스템콜 총 호출 수가 모델 예측에 있어 가장 결정적인 영향을

www.kci.go.kr

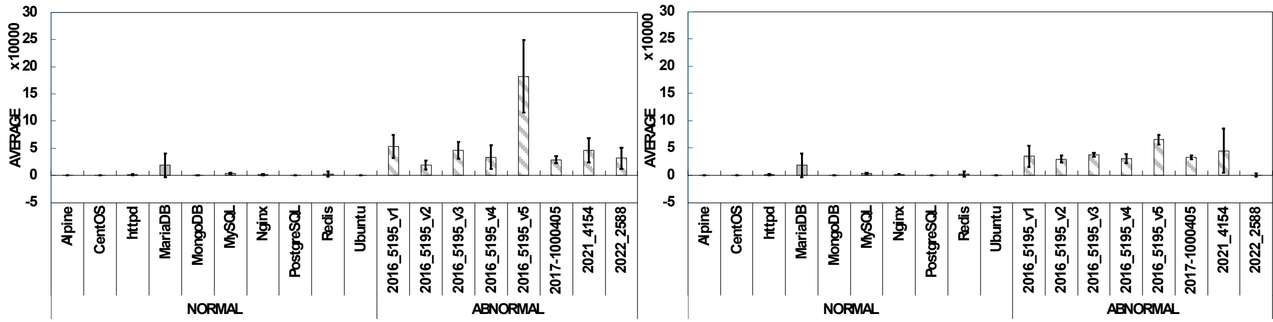


Fig. 4. SYSCALL_TOTAL_COUNT, Left with CVE fix, Right without CVE fix

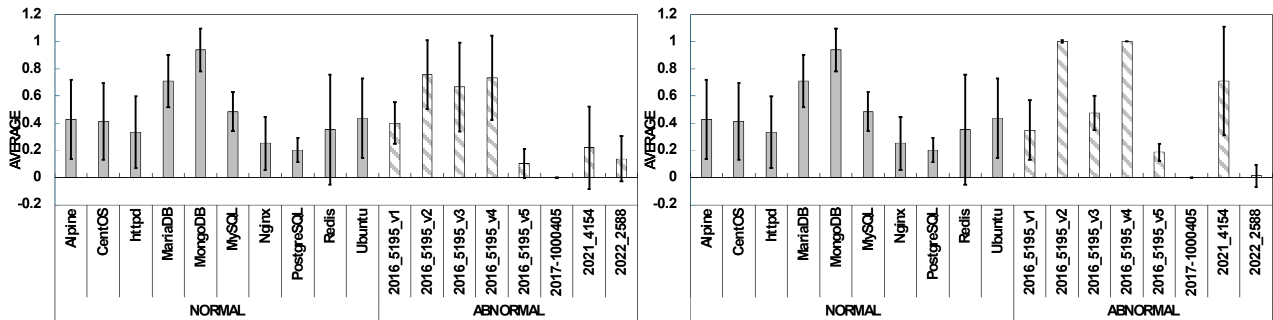


Fig. 5. SYSCALL_KIND_SIMILAR, Left with CVE fix, Right without CVE fix

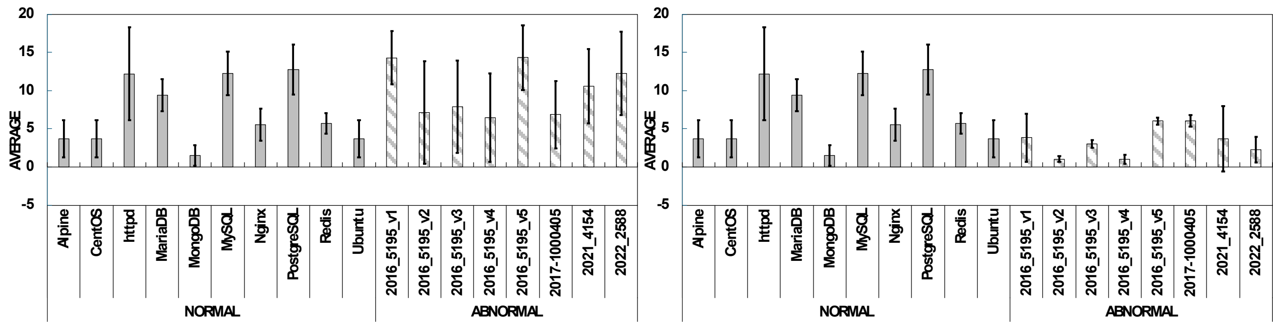


Fig. 6. SYSCALL_KIND, Left with CVE fix, Right without CVE fix

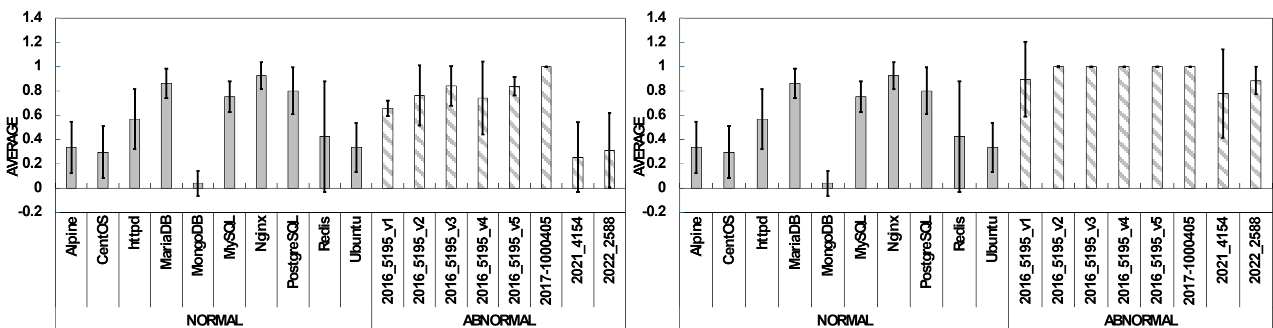


Fig. 7. SYSCALL_ARGUMENT_SIMILAR, Left with CVE fix, Right without CVE fix

미치는 요소임을 시사한다.

### 4. Overhead

프로그램 동작에서 나타나는 오버헤드를 확인하기 위해 Phoronix Test Suite[23]를 사용하였다. Phoronix Test Suite는 일련의 성능 벤치마크들을 제시하는 포괄적인 벤치마킹 플랫폼으로, 그중에서 OSBench와 Apache HTTP 테스트를 진행하였다. 테스트는 리눅스 5.10.90 커널을 실행하는 x86_64 아키텍처 리눅스 가상 머신에서 진행되었다. 가상 머신은 2.50GHz에서 실행되는 4개의 코어와

www.kci.go.kr

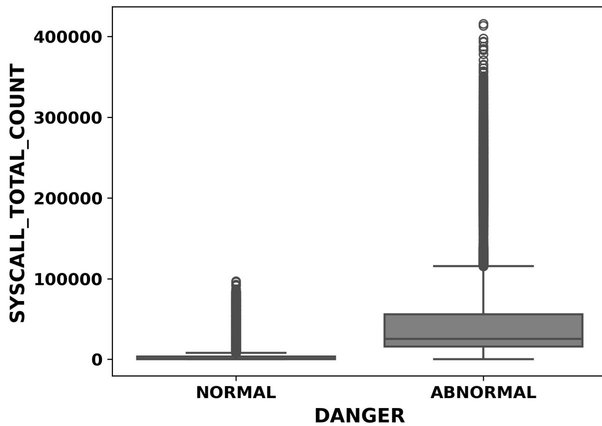


Fig. 8. SYSCALL_TOTAL_COUNT Dataset Graph

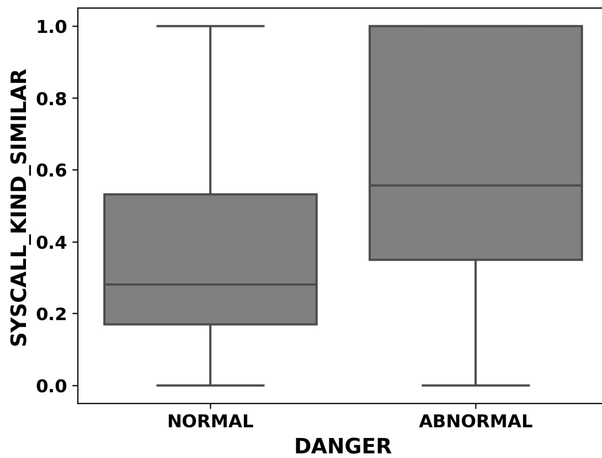


Fig. 9. SYSCALL_KIND_SIMILAR Dataset Graph

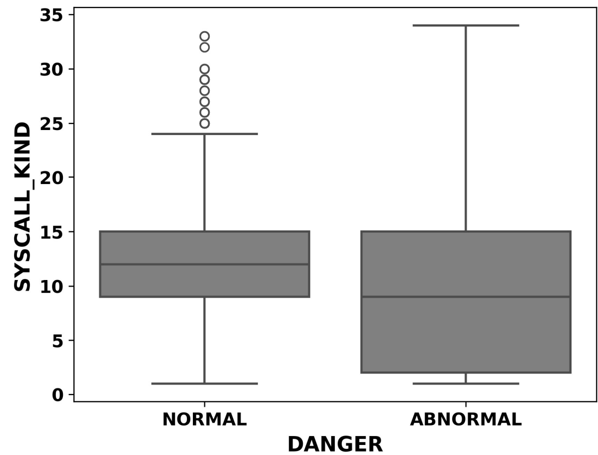


Fig. 10. SYSCALL_KIND Dataset Graph

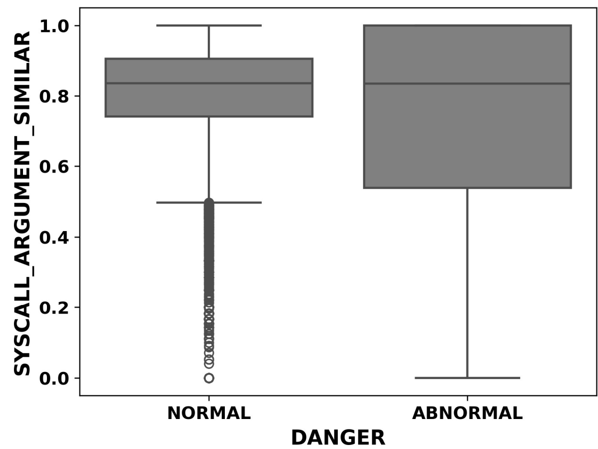


Fig. 11. SYSCALL_ARGUMENT_SIMILAR Dataset Graph

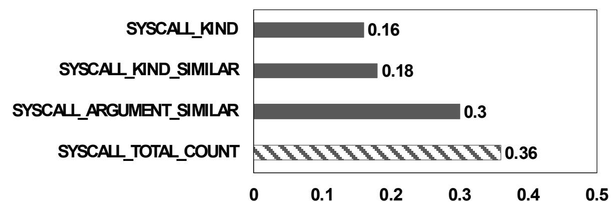


Fig. 12. AdaBoost Feature Importance

8GB의 RAM을 가지고 있다. 이와 관련된 오버헤드 측정 결과는 Table. 3에 자세히 기술되어 있다.

Phoronix OSBench benchmarking suite는 운영 체제의 다양한 기능에 대한 성능을 측정하기 위해 설계된 일련의 테스트를 제공한다. 성능 평가로서 Create Files, Create Threads, Launch Programs, Create Processes, Memory Allocations에 대해 평가한다. OSBench 벤치마크의 결과는 제안하는 프로그램이 실행되는 시스템에 대해 모든 테스트에서 평균 약 8% 정도의 오버헤드를 발생시키는 것을 보인다. 특히 Launch Programs에서는 약 26.09%의 가장 큰 성능 차이를 보인다. 반면, Memory Allocations에서는 제안하는 프로그램을 실행하였을 때의 시스템이 약간 더 나은 성능을 보여주지만 1 표준 편차 미만으로 분리되어 있으므로 이 테스트는 본질적으로 비슷한 수준으로 볼 수 있다.

## VI CONCLUSION

본 연구에서는 컨테이너 환경의 보안 강화를 위한 새로운 접근 방식을 제시하였다. 기존의 방법들은 파일에 대한 경쟁 상태 기반 공격만을 감지하거나, 커널을 수정해주어야 하는 등의 제한 사항을 가지나, 본 연구에서는 이를 극복하기 위해 시스템콜을 활용하여 경쟁 상태 기반 공격을 감지하는 방법으로 해결하고자 하였다. 공격을 감지하기 위해 경쟁 상태 기반 공격들이 호출하는 시스템콜 특징을

www.kci.go.kr

eBPF를 이용하여 확보하고, 확보한 시스템콜 데이터를 바탕으로 AdaBoost 모델에 학습시켰다. 학습된 모델은 테스트 데이터를 이용한 분류에서 Precision 99.97%, Recall 99.75% 그리고 F1-score 99.86%를 달성하였으며, eBPF를 사용함으로써 발생하는 오버헤드는 약 8% 정도로 나타났다. 향후 연구에서는 본 연구에서 제시한 방법을 바탕으로, 경쟁 상태 기반 감지뿐만 아니라 경쟁 상태 기반 공격의 방어 메커니즘을 개발함으로써 컨테이너 환경의 보안성을 더욱더 강화할 수 있는 방안을 모색할 예정이다.

## ACKNOWLEDGEMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No.NRF-2021R1A5A1021944).

## REFERENCES

[1] "CNCF Annual Survey ," 2022. [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2022/.

[2] "Dirty COW," [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2016-5195/.

[3] Z. Lin, Y. Wu and X. Xing, "DirtyCred: Escalating Privilege in Linux Kernel," in In Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, 2022.

[4] "Docker container escape using the Dirty COW vulnerability," [Online]. Available: https://blog.paranoidsoftware.com/dirty-cow-cve-2016-5195-docker-container-escape/.

[5] "A new method for container escape using file-based DirtyCred," 2023. [Online]. Available: https://starlabs.sg/blog/2023/07-a-new-method-for-container-escape-using-file-based-dirtycred/.

[6] B. Barth, "New cryptominer seeks out root permissions on Linux," [Online]. Available: https://www.scmagazine.com/news/new-cryptominer-seeks-out-root-permissions-on-linux-machines.

[7] M. Belair, S. Laniepce and J. Menaud, "SNAPPY: programmable kernel-level policies for containers," in In Proceedings of the 36th Annual ACM Symposium on Applied Computing, 2021.

[8] N. DeMarinis, K. Williams-King, D. Jin, R. Fonseca and V. P. Kemerlis, "Sysfilter: Automated system call filtering for commodity software," in In 23rd International Symposium on Research in Attacks, Intrusions and Defenses(RAID 2020), 2020.

[9] W. Findlay, A. Somayaji and D. Barrera, "Bpfbox: Simple precise process confinement with ebpf," in In Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop, 2020.

[10] S. Ghavamnia, T. Palit, A. Benameur and M. Polychronakis, "Confine: Automated system call policy generation for container attack surface reduction," in In 23rd International Symposium on Research in Attacks, ntrusions and Defenses (RAID 2020), 2020.

[11] S. Lim, B. Stelea, X. Han and T. Pasquier, "Secure namespaced kernel audit for containers," in In Proceedings of the ACM Symposium on Cloud Computing, 2021.

[12] F. Loukidis-Andreou, I. Giannakopoulos, K. Doka and N. Koziris, "Docker-sec: A fully automated container security enhancement mechanism," in In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018.

[13] Y. Sun, D. Safford, M. Zohar, D. Pendarakis, Z. Gu and T. Jaeger, "Security namespace: making linux security frameworks," in In 27th USENIX Security Symposium (USENIX Security 18), 2018.

[14] J. Johansen, "About AppArmor," 2019. [Online]. Available: https://gitlab.com/apparmor/apparmor/-/wikis/About.

[15] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka and D. Popa, "Firecracker: Lightweight virtualization for serverless applications," in In 17th USENIX symposium on networked systems design and implementation (NSDI 20), 2020.

[16] "gVisor," [Online]. Available: https://gvisor.dev/.

[17] A. Randazzo and I. Tinnirello, "Kata containers: An emerging architecture for enabling mec services in fast and secure way," in In 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), 2019.

[18] L. Rice, Container security: Fundamental technology concepts that protect containerized applications, O'Reilly Media, Inc.: 2020,

[19] S. Kim, B. Kim and D. Lee, "Prof-gen: Practical study on system call whitelist generation for container attack surface reduction," in In 2021 IEEE 14th International Conference on Cloud Computing (CLOUD), 2021.

[20] "eBPF," [Online]. Available: https://ebpf.io/.

[21] "Berkeley Packet Filter," [Online]. Available: https://en.wikipedia.org/wiki/Berkeley_Packet_Filter.

[22] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," in USENIX winter, 1993.

[23] M. Tippett and M. Larabel, "Phoronix Test Suite," [Online]. Available: https://www.phoronix-test-suite.com/.

www.kci.go.kr

## Authors

Hyeonseok Shin received the B.S. degree in Computer Science and Engineering from Kyungpook National University, Daegu, Korea in 2023 and currently doing a M.S course from 2023 in Kyungpook National University. He is interested in cloud computing, and distributed systems.

Minjung Jo received the B.S. degree in Software Engineering from Cyber Univercity of Korea, Korea, in 2023. She is currently doing a M.S Course in the Department of Computer Science and Engineering at Kyungpook National University from 2023. She is interested in cloud computing, and distributed systems.

Hosang Yoo received the B.S. degree in Computer Engineering from Kyungnam University, Korea in 2023 and currently doing a M.S course from 2023 in Kyungpook National University. He is interested in cloud computing, and distributed systems.

Yongwon Lee received the B.S. degree in Computer Software from Daegu University, Korea, in 2022. He is currently doing a M.S Course in the Department of Computer Science and Engineering at Kyungpook National University from 2020. He is interested in cloud computing, and distributed systems.

Byungchul Tak received his B.S. degree from Yonsei University in 2000, M.S. from KAIST in 2003 and Ph.D. degrees in Computer Science and Engineering from the Pennsylvania State University at University. Park in 2012. Dr. Tak joined the faculty of the School of Computer Science and Engineering at Kyungpook National University, Dague, Korea, in 2017. He is currently an Associate Professor. His research interests are in cloud computing, distributed systems, operating system and big data analytics.

www.kci.go.kr